NOTICE: When government or other drawings, specifications or other data are used for any purpose other than in connection with a definitely related government procurement operation, the U. S. Government thereby incurs no responsibility, nor any obligation whatsoever; and the fact that the Government may have formulated, furnished, or in any way supplied the said drawings, specifications, or other data is not to be regarded by implication or otherwise as in any manner licensing the holder or any other person or corporation, or conveying any rights or permission to manufacture, use or sell any patented invention that may in any way be related thereto.

PROJECT

lightning

62-3-1

Copy No. 118

INTERIM RESEARCH REPORT NO. 11A

for

HIGH-SPEED DATA PROCESSOR SYSTEM RESEARCH

Project LIGHTNING

APPENDIX II

SYSTEM STUDY REPORT NUMBER 7

This report covers the period of
June 1, 1961 to August 31, 1961

Prepared for
DEPARTMENT OF THE NAVY
Bureau of Ships, Electronics Division
Washington 25, D. C.
Contract Number - NObsr 77523, February 2, 1959
and Amendment Number 8 — June 27, 1960

Prepared by
Engineering Department
Electronic Data Processing Division
Industrial Electronic Products
RADIO CORPORATION OF AMERICA
Camden, New Jersey

PROJECT LIGHTNING

System Study Report Number 7

by

M. Sendrow

TABLE OF CONTENTS

APPENDIX II

SUMMARY

~~This Appendix details~~ the design of a computing system proposed under different assumptions from the ones previously proposed under this title is detailed. The structure of the machine is outlined; the registers, sequencing, addressing, instruction repertoire, and the elaborate input-output system are described. Many questions of detail remain to be worked out.

A. INTRODUCTION

This report is somewhat different from the previous ones under this title authored by P. Warburton, in that the assumption is made that the present technological difficulties in the use of tunnel diodes for computer elements will be overcome in due course, and it will then be possible to build a computing machine of reasonable size. The machine herein proposed, called Minotaur I, is a large, binary data-processor, with extensive input-output facilities. The arguments for and against proposing such a machine will not be repeated here as they have been thoroughly documented in the previous LIGHTNING reports. One purpose in proposing a machine of this size now is to try to determine what is the power of such a machine at LIGHTNING speed. It may be that the increased power is great enough that it should still be aimed at, even at the present technological "state-of-the-art." Finally, the device proposed herein should not be thought of as a "finished design," but rather thought of as the start of a design cycle. Many areas are hazy due to hardware considerations, and some due to programming considerations. Detail is presented where possible, and general outlines are presented where detail is not possible.

B. DEFINITIONS

Certain definitions are presented below, while others are given in their associated section where their meaning will be clearer.

Symbol	Definition
(y)	Contents of y
x (j - k)	Bit positions j to k of word x
Acc (n)	Bit position n of the Accumulator (similar for other registers)
ACC(s)	The contents of the Accumulator shifted
Y(i)	The initial contents of location Y
Acc(i)	The initial contents of the Accumulator (similar for other registers)
Y(f)	The final contents of location Y
ACC(f)	The final contents of the Accumulator
NI	Next Instruction

C. WORD STRUCTURE

1. Instruction Word

A word in Minotaur I consists of 60 bits, numbered 59 to 0, left to right, respectively. An instruction contains an op symbol of 12 bits; 2 addresses called U and V addresses, each 18 bits; and 2 B-box addresses of 6 bits each. These 60 bits are arranged in the order listed below and as illustrated in Figure II-1.

Op Symbol	Bits 59 to 48
B (U)	Bits 47 to 42
U Address	Bits 41 to 24
B (V)	Bits 23 to 18
V Address	Bits 17 to 0

B (U) and B (V) symbolize the U address B-box and the V Address B-Box, respectively. Bit 48 is the op symbol repeat bit (as explained in the section on the repeat mode), and bits 41, 40, and 17, 16 are the immediate and indirect bits for the U and V addresses, respectively. Immediate and indirect addressing will be explained in the section on sequencing.

2. B-Box Word

B-boxes are of full word length and are contained in a special, small, high-speed memory. There will be approximately 64 B-boxes, some of which will be the registers of the machine. This will be explained more fully below. The structure of the B-box word will be as follows and as shown in Figure II-2.

Count	Bits 59 to 42
Increment	Bits 41 to 24
Jump increment	Bits 23 to 18
Modifier	Bits 17 to 0

In the non-repeated mode when a B-box number other than zero is used in B (U) or B (V), the modifier is added to the respective address in the proper register. The count, increment, and jump increment are used in the repeat mode. Note that when the modifier is added to the address, no overflow is permitted into the indirect and immediate bits of the address, i.e., the addition is only on 16 bits. Also, the addition is 2's complement.

3. Input-Output Word

These have a slightly different structure from the other words and are discussed in the section on input-output.

4. Data Word

Data may take any form in the word or part of the word, i.e., alpha-numeric, BCD, 1's complement, etc.

D. MEMORY STRUCTURE

1. B-Box Memory

As mentioned above, there will be a separate, small, high-speed memory of 64 words of 60 bits each. It is hoped that the speed of this memory will be at least twice that of the regular high-speed memory.

2. High-Speed Memory

The high-speed memory will consist of 4096 words of 60 bits each, with associated addressing circuitry and a regeneration register. For technical reasons, a memory may be built, with a 64-bit word, in which case the extra bits will be used for error detection and correction.

3. Intermediate-Speed Memory

There will have to be a back-up memory, somewhat slower than the tunnel diode high-speed memory. Whether this will be transistor driven fast cores, tunnel diode driven plates, thin films, etc., is undetermined at this time. Also, the amount and type of buffering between the high-speed and intermediate-speed memory is unknown at this time. Hopefully, some method of simultaneous transfer can be arranged with a minimum of high-speed memory time used in the transfer. For instance, it might be possible to arrange four banks of intermediate-speed memory to correspond to four banks of tunnel diode memory, so that if the ratio of both memories is 5 to 1, four words can be transferred every five cycles, effectively interrupting only once in 20 cycles. However, details on this scheme will have to await technical developments.

E. ADDRESSING STRUCTURE

The high-speed memory is addressed with the U and V addresses of instructions. There will be some standard addresses also in the high-speed memory, for instance, for the trace jump. The B-box memory will be doubly addressable in the following sense. The B-box memory can be addressed by using B (U) and B (V) in instructions. However, the B-box memory logically replaces the first part of the regular high-speed memory from an address point of view. In this way, if one of the first 64 locations is addressed by a U or V address, the B-box memory contents will be used. The B-box memory than becomes a higher-speed adjunct to the regular high-speed memory. In addition, some of the control registers will also be B-boxes, replacing the beginning of the B-box memory. This makes those registers triply addressable. For instance, the accumulator can be addressed as a high-speed memory location by using a U or V address; as a B-box by using B (U) or B (V); or implicitly, as in an add instruction. This method complicates the address decoders in the program control unit to some extent, but provides a great deal of flexibility in use of the registers and B-boxes. It will be noted that since the B-boxes are of full word length and can be addressed like the regular high-speed memory, short loops can be placed entirely in the B-box memory and can be executed very rapidly. This, of course, presupposes a certain unusual type of processing which does arise on certain extended problems.

It has not yet been decided whether the intermediate-speed memory will be "word addressable" or "block addressable". That decision will have to await firmer characteristics of the memory itself.

F. REGISTERS

The registers and their interconnections are shown in Figure II-3 and are described below. The abbreviations which follow each register designation will be used in the remainder of this Appendix.

(1) Accumulator (Acc) — a 60-bit register used in arithmetic instructions, logical instructions, and some transfer instructions. Includes a shift and borrow network and magnitude determining logic.

(2) Q Register (Q) — a 60-bit register used in multiply, divide, logical instructions, and some transfer instructions. Includes a shift network and some magnitude determining logic.

(3) X Register (X) — a 60-bit register used for most register-to-register transfers; register-to-memory, memory-to-register transfers; memory-to-memory transfers; and in logical instructions. Also, used in transferring information to and from the input-output control.

(4) Regeneration Register (RR) — a 60-bit register used for cycling information into and out of the memory. There may be two of these: one for the high-speed memory and one for the intermediate-speed memory. Hereinafter, when RR is used, it refers to the high-speed memory.

(5) Program Address Register (PAR) — a 16-bit register that holds the address of the instruction currently being executed.

(6) Memory Address Register (MAR) — a 16-bit register that holds the address of the word currently being read from or written into the memory. It is used to set the address drivers of the memory.

(7) Modifier Adder (MAD) — a 16-bit register that can add two numbers using 2's complement arithmetic, add 1 to a number, test for zero, subtract 1 from a number, and do a limited number of other functions. It is used to advance the PAR, add the modifier to the base address, add the increment to the modifier in the repeat mode, and do similar functions for the control registers of the machine. This will be seen more fully in the examples of the subcommands and sequencing described later in a separate section.

(8) Translator (XTR) — a 6-to 16-bit decoding network used to supply memory addresses when given the B-box number, in addition to other functions.

(9) Instruction Control Register (ICR) — a generic name used to designate all the separate registers that hold the various parts of an instruction. The ICR is 60 bits long and consists of the following registers:

(a) Op Code Register — 12 bits, including the repeat control FF.

(b) U Address B-Box Designator (BUR) — 6 bits, used to hold B (U).

(c) U Address Register (UAR) — 18 bits, used to hold the U address as it comes from the memory before B-boxing, etc.

(d) V Address B-Box Designator (BVR) — 6 bits, used to hold B (V).

(e) V Address Register (VAR) — 18 bits, used to hold the V address as it comes from the memory before B-boxing, etc.

(10) U Address B-Box Register (UBR) — a generic name used to designate all the separate registers that hold the various parts of a B-box word as it comes from the memory and after modification during repeat mode execution. The UBR is 60 bits long, and consists of the following registers:

(a) U Address Count Register (UCTR) — 18 bits,holds the U address count.

(b) U Address Increment Register (UIR) — 18 bits, holds the U modifier increment.

(c) U Jump Increment Register (UJR) — 6 bits, holds the increment used at the termination of a repeat sequence.

(d) U Address Modifier Register (UMR) — 18 bits, holds the modifier portion of the B-box word.

(11) V Address B-Box Register (VBR) — similar to the UBR, except it applies to the V address instead of the U Address. Consists of the V Address Count Register (VCTR), V Address Increment Register (VIR), V Jump Increment Register (VJR), and the V Address Modifier Register (VMR).

(12) Effective U Address Register (U EFF) — an 18 bit register that holds the sum of the UAR and the UMR in B-boxed instructions, holds each indirect address in an indirect chain, etc., and has facility for sensing immediate and indirect bits. This register functions as the logical UAR considering B-boxing, indirect addressing, etc. Its role will be made clearer in the examples of sequencing explained in a separate section.

(13) Effective V Address Register (V EFF) — similar to the U EFF, except it applies to the V Address.

(14) Jump Register — a 16-bit addressable register used to hold a jump address for certain index jump instructions.

(15) I/0 Exchange Register (I/0 X) — a 60-bit register used to transfer information between the input-output devices and the memory.

(16) I/0 Interrupt Register — length unknown at present, accepts signals from external devices concerning their status.

(17) I/0 Mask Register — length unknown at present, used to determine if a signal in the interrupt register should be permitted to interrupt the computer.

(18) Real Time Clock — 48 bits, functions as an immediately accessible I/0 device, and can be set, stored, etc., by program.

It may be necessary to add another transfer register (Z Reg) later depending on the mechanization of certain functions, but it will be omitted for the present.

Of the registers described above, the following will be doubly addressable, as B-boxes and as memory: Acc, Q, PAR, UBR, VBR. The following will also be addressable as memory: I/0 Mask, I/0 Interrupt, perhaps some others. There may have to be special provision to give programmers access to the U EFF and V EFF.

G. TIMING ASSUMPTIONS

In Section H. Sequencing, the following assumptions have been made about times for various operations.

(1) High-Speed Memory Cycle Time — 20 nanoseconds (ns)

(2) Register-to-Register Transfer Time — 2 ns.

(3) B-Box Memory Cycle Time — 6 ns.

(4) Acc Add Time, Total — 10 ns.

(5) MAD Add Time, Total — 6 ns.

These assumptions may not turn out to be extremely accurate; however, at the present time, they seem reasonable.

H. NON-REPEAT SEQUENCING

This section details the sequencing of an instruction in the non-repeated mode. The procedures for reading an instruction from the memory to the control registers, checking B-box addresses, reading B-box contents from the memory to the control registers, adding modifiers to base addresses, following an indirect chain to an immediate or direct address, and similar operations (see Figure II-4) will be called

"Euprosodopoesis"* from the Greek, "to make easily accessible". In general, for the non-repeated mode euprosodopoeticizing on Minotaur I will consist of the following: the PAR will be advanced (in MAD); the instruction will be read from the memory to the ICR through the regeneration register and the X register; B (U) and B (V) will be checked for zero (in MAD), and if they are both zero, the U and V addresses will be transferred from UAR and VAR to U EFF and V EFF, respectively, to be checked for immediate and indirect addresses. (Note: an immediate address is one which is the operand; a direct address is one which is the address of the operand; and an indirect address is one which is the address of the address ... of the operand.) An immediate address has meaning only when it refers to an operand to be retrieved, not when it refers to an operand to be stored. If there are neither indirect nor immediate addresses, the U and V operands are read out of memory in sequence and the instruction performed.

In order to clarify the above sequencing, the following simple example is presented in detail. For this example assume the instruction to be executed is a Replace Add, that there are no B-boxes called for, that both U and V are direct, and that the timing is that specified in Section G.

Example:

Replace Add (U) + (V) → U (each step represents 2 ns)

	Operations	Register Affected	Contents
(1)	Trigger MAD	MAD	+1
(2)	PAR to MAD	MAD	PAR + 1
(3)	MAD to PAR and MAR Clear MAD	PAR and MAR MAD	PAR + 1 Clear
(4)	Read Memory (instruction)		
(5)	Delay, Memory		
(6)	Delay, Memory		
(7)	Delay, Memory		
(8)	Word appears in RR	RR	(PAR + 1)
(9)	RR to X Reg	X	(PAR + 1)
(10)	X Reg to ICR	ICR	(PAR + 1) (the instruction)

* Suggested by R. Herwitt, RCA EDPD.

Operations	Register Affected	Contents
(11) B(U) to XTR	XTR	B (U)
(12) XTR to MAD	MAD	O
(13) UAR to U EFF	U EFF	U Address
(14) U EFF to MAR	MAR	U Address (U Add)
(15) Read Memory (U Add)		
(16) B(V) to XTR	XTR	B (V)
(17) XTR to MAD	MAD	O
(18) VAR to V EFF	V EFF	V Address (V Add)
(19) Word appears in RR	RR	U Operand
(20) RR to X Reg Clear Acc	X Reg	U Operand
(21) X Reg to Acc	Acc	U Operand
(22) Delay, Memory		
(23) Delay, Memory		
(24) V EFF to MAR	MAR	V Address
(25) Read Memory (V Add)		
(26) Delay, Memory		
(27) Delay, Memory		
(28) Delay, Memory		
(29) Word appears in RR	RR	V Operand
(30) RR to X Reg.	X Reg.	V Operand
(31) X Reg. to Acc.	Acc.	U Operand + V Operand
(32) Delay, Memory and Acc.		
(33) Delay, Memory and Acc.		
(34) Delay, Memory and Acc.		

Operations	Register Affected	Contents
(35) U EFF to MAR (Delay, Acc.)	MAR	U Address
(36) Acc to X Reg.	X Reg.	U Operand + V Operand
(37) Write Memory		
(38) X Reg to RR	RR	U Operand + V Operand
(39) Delay, Memory		
(40) Delay, Memory		
(41) Delay, Memory		
(42) Delay, Memory		
(43) Delay, Memory		
(44) Delay, Memory		
(45) Delay, Memory		
(46) Delay, Memory		

Total time — 92 ns.

It will be noticed that some operations were overlapped, for instance, 16, 17, and 18 were overlapped with a memory delay. This becomes even more apparent in a more complicated example.

It was mentioned before that U EFF and V EFF were the registers that hold the U Add and V Add for indirect chaining and immediate addressing. This operates in the following manner. After the U and V addresses have been modified (if required) by the B-box modifier, they are sent to U EFF and V EFF. Under local control, the indirect bit is checked for a "1". If it is a "1", then the address is sent from U EFF or V EFF to the MAR, and the next link in the chain replaces the previous link in U EFF or V EFF. The end result of the chain is either a direct or immediate address, in which case the normal euprosodopoeticizing sequence continues. Note that during B-boxing only 16 bits of the U or V address participate in the addition; in indirect addressing, all 18 bits of the new address replace the old address in U EFF or V EFF. This scheme gives control of the level of indirect addressing to the address called for, itself, and not to the address in the instruction or the B-box constant. There is a danger here, of course, of getting into an "infinite, indirect loop." Some special provision may have to be made to check for that possibility. The convention is made that a "1" bit in the indirect or immediate positions signifies an indirect or immediate address, and a "O" bit in these positions signifies a direct address. Through error, both the indirect and immediate bits may be "1" 's in an address; a priority scheme will be chosen later to handle this exigency. Those registers which have memory addresses, like PAR, and which could be used as a link in an indirect chain, will have their indirect bits forced to "0" and therefore these registers must be the last link in the chain. It has not yet been decided positively that the immediate bit will also be forced to "0", but it seems likely.

If the first address to show up in U EFF or V EFF is an immediate address, then it participates in the instruction as an operand, provided that an immediate address is permitted in that particular instruction. In many cases the operand will have to be expanded to the full 60 bits by extending the most significant bit, i.e., the 40'th or 16'th bit for U and V, respectively, not the immediate bit or indirect bit. (The 40'th bit is number 39, and the 16'th bit is number 15.) Exactly where the address will be expanded is not yet known; however, some register or network will be chosen later when a more detailed examination of the instruction mechanization has been completed.

With these further explanations, it is now possible to take a more complicated example than the one chosen before, and examine in detail the subcommands necessary to execute an instruction. The same instruction as before will be chosen, Replace Add, but B-boxing and indirect addressing will be illustrated this time. The same timing as before will be assumed.

Example:

Replace Add (U) + (V) → U . B(U) & B(V) $\neq$ O, U and V each have one indirect address which is stored in the high-speed memory (access time, 20 ns.) and (B(U)) & (B(V)) are stored in the B-box memory.

Operations	Register Affected	Contents
(1) Trigger MAD	MAD	+1
(2) PAR TO MAD	MAD	PAR + 1
(3) MAD to PAR and MAR Clear MAD	PAR and MAR MAD	PAR + 1 Clear
(4) Read Memory (instruction)		
(5) Delay, Memory		
(6) Delay, Memory		
(7) Delay, Memory		
(8) Word Appears in RR	RR	(PAR + 1)
(9) RR to X	X	(PAR + 1)

Operations	Register Affected	Contents
(10) X to ICR	ICR	(PAR + 1) (the Instruction)
(11) B(U) to XTR	XTR	B (U) extended
(12) XTR to MAD	MAD	B (U) extended
(13) MAD to MAR	MAR	B (U) extended
(14) Read Memory (B-Box) - (U)*		
(15) B(V) to XTR	XTR	B (V) entended
(16) XTR to MAD - (V)	MAD	B (V) extended
(17) RR (B-Box) to X-(U)	X	(B (U))
(18) MAD to MAR - (V)	MAR	B (V) entended
(19) Read Memory (B-Box) - (V)		
(20) X to UBR - (U)	UBR (UCTR, UIR, UJR, UMR).	U CT, U Add Increment, U Jump Increment, U Modifier
(21) UMR to MAD - (U)	MAD	U Modifier
(22) UAR to MAD - (U) RR(B-Box) to X - (V)	MAD X	U Modifier + U Add = U(M) (B(V))
(23) MAD to U EFF and MAR - (U) X to VBR	U EFF and MAR VBR	U(M) (indirect U Address) U CT, etc.
(24) Read Memory (U Indirect)		
(25) VMR to MAD - (V)	MAD	V Modifier
(26) VAR to MAD - (V)	MAD	V Modifier + V Add = V(M)
(27) MAD to V EFF - (V)	V EFF	V(M) (indirect V Address)
(28) Word appears in RR - (U) (U direct address)	RR	U(M)'

* (U) means operating on the U addresses, (V) means operating on the V addresses, and are included in the example for clarity only.

Operations	Register Affected	Contents
(29) RR to X - (U)	X	U (M)'
(30) X(41 - 24) to U EFF - (U)	U EFF	U (M)'
(31) MAD to MAR - (V)	MAR	V (M)
(32) U EFF to MAD - (U)	MAD	U (M)'
(33) Delay, Memory		
(34) Read Memory (V indirect)		
(35) Delay, Memory		
(36) Delay, Memory		
(37) Delay, Memory		
(38) Word appears in RR - (V) (V direct address)	RR	V (M)'
(39) RR to X - (V)	X	V (M)'
(40) X(17 - 0) to V EFF - (V)	V EFF	V (M)'
(41) MAD to MAR - (U)	MAR	U (M)'
(42) Delay, Memory		
(43) Delay, Memory		
(44) Read Memory (U Operand)		
(45) V EFF to MAD - (V)	MAD	V (M)'
(46) Delay, Memory		
(47) Delay, Memory		
(48) Word appears in RR - (U) (U operand)	RR	U Operand
(49) RR to X - (U) Clear Acc	X Acc	U Operand Zero

	Operations	Register Affected	Contents
(50)	X to Acc - (U)	Acc	U Operand
(51)	Delay, Memory		
(52)	Delay, Memory		
(53)	MAD to MAR - (V)	MAR	V (M)'
(54)	Read Memory (V operand)		
(55)	Delay, Memory		
(56)	Delay, Memory		
(57)	Delay, Memory		
(58)	Word appears in RR - (V) (V operand)	RR	V Operand
(59)	RR to X - (V)	X	V Operand
(60)	X to Acc	Acc	U Operand + V Operand = U Result
(61)	Delay, Memory and Acc		
(62)	Delay, Memory and Acc		
(63)	Delay, Memory and Acc		
(64)	Delay, Acc U EFF to MAR - (U)	MAR	U (M)'
(65)	Acc to X - (U)	X	U Result
(66)	Write Memory (U)		
(67)	X to RR - (U)	RR	U Result
(68)	Delay, Memory		
(69)	Delay, Memory		
(70)	Delay, Memory		

Operations	Register Affected	Contents
(71) Delay, Memory		
(72) Delay, Memory		
(73) Delay, Memory		
(74) Delay, Memory		
(75) Delay, Memory		

Total Time, 150 ns.

As can be seen, the euprosodopoeticizing can become quite extensive. Some of the operations that were overlapped were # 15, 16, 20, 21, 25, 26, 27, 31, 32, and several others.

It will be noticed that most of the delays occurred because of the memory, for instance 51 and 52; however, occasionally, some time was taken up in register transfers that couldn't be done in parallel with memory accesses, for instance 22 and 23. This balance will change, of course, with slightly different time ratios between memory and register transfers, and particularly with slightly different estimates of the B-box memory time. The example given was not to decide once and for all the timing of the various subcommands, but to illustrate the uses of the various registers and to show the operation of B-boxing and indirect addressing, among other things.

I. REPEAT SEQUENCING

The purpose of the repeat mode is to repeat one specified instruction a number of times. (That statement usually causes acrimonious debate concerning the value of a function that is so restricted. The question will not be argued here).

Bit 48 is the repeat bit in those instructions which can be repeated. It is hoped that most instructions will be repeatable. If bit 48 is "1", the repeat mode is initiated. This mode operates as follows: the normal euprosodopoeticizing is followed up through the B-boxing, then the U address increment is added to U modifier (for the next cycle), one is subtracted from U Ct, and U Ct is tested for negative. U Ct negative signals the end of repeat. (This may be changed to "0" depending on the functions of MAD). If U Ct is not negative, then the V values are processed in a like manner. The instruction is then executed and another cycle is initiated, except that the instruction and B-boxes already in the registers are used instead of reading both out of the memory. It will be noticed that on succeeding cycles, when the incremented modifier is added to U or V, the U or V addresses come from UAR or VAR, but the results always go into U EFF and V EFF. Thus, the U address increases by the increment on each succeeding cycle, i.e., the successive effective U addresses are, U address + U modifier, U address + U modifier + U increment, U address + U modifier + 2 U increment, U address + U modifier + 3 U increment, etc. Where they apply, both immediate and indirect addressing may be used in the repeat mode at each cycle. In this manner, a list need not be stored in consecutive locations; the only requirement is that the addresses of the items in the list must be stored a fixed number of locations from each other (equal to the increment).

In those cases where it is desired to keep one or both of the U and V addresses fixed in the repeat mode, then the increment must be made zero. This is necessary since in the repeat mode the increment is always added to the modifier and the modifier is always added to the address. The modifier, of course, can be made zero at the programmer's option.

At the point during the cycling when either U Ct or V Ct is exhausted, the repeat mode is terminated by a self-relative jump in the following manner: U Jump Increment, or V Jump Increment, as the case may be, is sent to the XTR, and its most significant bit is extended to make the increment 16 bits long. XTR is transferred to MAD, then PAR is transferred to MAD, and the sum is sent to PAR and MAR to start the sequencing for the next instruction. Considering the triggering of MAD at the beginning of euprosodopoeticizing, a jump can be made up to 32 instructions forward, or 31 backward. Admittedly, this is not ideal, but represents a reasonable compromise between the ideal and the Procrustean Bed of fixed-word computers.

Recycling starts at the point where U Mod is added to U Add in MAD. With the increased processing that takes place during repeat sequencing, there is more of an opportunity to parallel operations. This can be seen in the following lengthy example of Replace Add, repeated twice, with U and V indirect addresses. The same timing as before is assumed.

A flow chart of the euprosodopoeticizing that is illustrated in the following example is given in Figure II-4. The chart is not strictly accurate in the sense that some procedures shown being done in sequence will actually be done partly or wholly in parallel. However, the chart does give some idea of the decision sequence involved in euprosodopoeticizing, under repeat and non-repeat mode, with and without B-boxes, indirect addresses, etc. Also, there is some idea given of the minimum amount of processing required.

Example:

Replace Add, (U) + (V) → U . B(U) & B(V) ≠ 0, U and V have one indirect address each per cycle, they are stored in the high-speed memory in each case, (B(U)) & (B(V)) are stored in the B-box memory, and U Ct = 2 (i.e., the instruction is to be repeated twice.)

Operations	Register Affected	Contents
(1) Trigger MAD	MAD	+1
(2) PAR to MAD	MAD	PAR + 1
(3) MAD to PAR and MAR Clear MAD	PAR and MAR MAD	PAR + 1 Clear

Operations	Register Affected	Contents
(4) Read Memory (Instruction)		
(5) Delay, Memory		
(6) Delay, Memory		
(7) Delay, Memory		
(8) Word appears in RR	RR	(PAR + 1)
(9) RR to X	X	(PAR + 1)
(10) X to ICR	ICR	(PAR + 1) (the instruction)
(11) B(U) to XTR	XTR	B (U) extended
(12) XTR to MAD	MAD	B (U) extended
(13) MAD to MAR	MAR	B (U) extended
(14) Read Memory (B-Box) - (U)		
(15) B(V) to XTR	XTR	B (V) extended
(16) XTR to MAD	MAD	B (V) extended
(17) MAD to MAR	MAR	B (V) extended
(18) RR(B-Box) to X - (U)	X	(B(U))
(19) Read Memory (B-Box) - (V)		
(20) X to UBR - (U)	UBR	U Ct, U Add Increment, U Jump Increment, U Modifier
(21) UMR to MAD - (U)	MAD	U Modifier
(22) UAR to MAD - (U) RR(B-Box) to X - (V)	MAD X	U Modifier + U Add = U(M) (B(V))
(23) X to VBR - (V)	VBR	V Ct, etc.
(24) Delay, MAD		

Operations	Register Affected	Contents
(25) MAD to U EFF - (U)	U EFF	U (M) (Indirect U address)
MAD to MAR - (U)	MAR	U (M)
(26) Read Memory (U Indirect)		
(27) UIR to MAD - (U)	MAD	U Add Increment
(28) UMR to MAD - (U)	MAD	U Add Increment + U Modifier = U Modifier'
(29) Delay, Memory and MAD		
(30) Delay, Memory and MAD		
(31) MAD to UMR - (U)	UMR	U Modifier'
Word appears in RR - (U) (U Direct Address)	RR	U (M)'
(32) RR to X - (U)	X	U (M)'
UCTR to MAD - (U)	MAD	U Ct
(33) MAD - 1 to MAD - (U)	MAD	U Ct - 1
(34) X to U EFF	U EFF	U (M)'
(35) Delay, Memory and MAD		
(36) MAD to UCTR - (U)	UCTR	U Ct - 1
VMR to MAD - (V)	MAD	V Modifier
(37) VAR to MAD - (V)	MAD	V Modifier + V Add = V (M)
(38) Delay, MAD		
(39) Delay, MAD		
(40) MAD to V EFF - (V)	V EFF	V (M)
MAD to MAR	MAR	V (M)
(41) Read Memory - (V) (V Indirect)		
VIR to MAD - (V)	MAD	V Add Increment
(42) VMR to MAD (V)	MAD	V Add Increment + V Modifier = V Modifier'

Operations	Register Affected	Contents
(43) Delay, Memory and MAD		
(44) Delay, Memory and MAD		
(45) MAD to VMR - (V)	VMR	V Modifier
Word appears in RR - (V)	RR	V (M)'
(V Direct Address)		
(46) RR to X - (V)	X	V (M)'
VCTR to MAD - (V)	MAD	V Ct
(47) MAD - 1 to MAD - (V)	MAD	V Ct - 1
X to V EFF	V EFF	V (M)'
(48) Delay, Memory and MAD		
(49) Delay, Memory and MAD		
(50) MAD to VCTR - (V)	VCTR	V Ct - 1
U EFF to MAR	MAR	U (M)'
(U Operand)		
(51) Read Memory (U Operand)		
(52) Delay, Memory		
(53) Delay, Memory		
(54) Delay, Memory		
(55) Word Appears in RR	RR	U Operand
(56) RR to X - (U)	X	U Operand
Clear Acc	Acc	Zero
(57) X to Acc - (U)	Acc	U Operand
(58) Delay, Memory		
(59) Delay, Memory		
(60) V EFF to MAR (V Operand)	MAR	V (M)'
(61) Read Memory (V Operand)		
(62) Delay, Memory		

Operations	Register Affected	Contents
(63) Delay, Memory		
(64) Delay, Memory		
(65) Word Appears in RR - (V)	RR	V Operand
(66) RR to X	X	V Operand
(67) X to Acc	Acc	U Operand + V Operand = U Result 1
(68) Delay, Memory and Acc		
(69) Delay, Memory and Acc		
(70) Delay, Memory and Acc		
(71) U EFF to MAR - (U) (U result)	MAR	U (M)'
(72) Acc to X - (U)	X	U Result 1
(73) Write Memory (U result 1)		
(74) X to RR (NOTE - START OF 2ND CYCLE)	RR	U Result 1
(75) UMR to MAD - (U)	MAD	U Modifier'
(76) UAR to MAD - (U)	MAD	U Modifier' + U Add = U (M1)
(77) Delay, Memory and MAD		
(78) Delay, Memory and MAD		
(79) MAD to U EFF - (U)	U EFF	U (M1)
(80) Delay, Memory		
(81) Delay, Memory		
(82) MAD to MAR (U Indirect)	MAR	U (M1)
(83) Read Memory (U Indirect) UIR to MAD	MAD	U Add Increment

Operations	Register Affected	Contents
(84) UMR to MAD	MAD	U Add Increment + U Modifier' = U Modifier''
(85) Delay, Memory and MAD		
(86) Delay, Memory and MAD		
(87) MAD to UMR	UMR	U Modifier ''
Word appears in RR	RR	U (M1)'
(U Direct)		
(88) RR to X - (U)	X	U (M1)'
UCTR to MAD - (U)	MAD	U Ct - 1
(89) MAD - 1 to MAD - (U)	MAD	U Ct - 2
X to U EFF	U EFF	U (M1)'
(90) Delay, Memory and MAD		
(91) Delay, Memory and MAD		
(92) MAD to UCTR - (U)	UCTR	U Ct - 2
VMR to MAD - (V)	MAD	V Modifier'
(93) VAR to MAD	MAD	V Modifier + V Add = V (M1)
(94) Delay, MAD		
(95) Delay, MAD		
(96) MAD to V EFF	V EFF	V (M1)
MAD to MAR	MAR	V (M1)
(97) Read Memory (V Indirect)		
VIR to MAD	MAD	V Add Increment
(98) VMR to MAD	MAD	V Add Increment + V Modifier' = V Modifier''
(99) Delay, Memory and MAD		
(100) Delay, Memory and MAD		
(101) MAD to VMR	VMR	V Modifier ''
Word appears in RR	RR	V (M1)'
(V Direct)		

Operations	Register Affected	Contents
(102) RR to X - (V) VCTR to MAD	X MAD	V (M1)' V Ct - 1
(103) MAD - 1 to MAD - (V) X to V EFF - (V)	MAD V EFF	V Ct - 2 V (M1)'
(104) Delay, Memory and MAD		
(105) Delay, Memory and MAD		
(106) MAD to VCTR -(V) U EFF to MAD - (U)	VCTR MAD	V Ct - 2 U (M1)'
(107) Read Memory (U Operand)		
(108) Delay, Memory		
(109) Delay, Memory		
(110) Delay, Memory		
(111) Word appears in RR	RR	U Operand
(112) RR to X - (U) Clear Acc	RR Acc	U Operand Zero
(113) X to Acc - (U)	Acc	U Operand
(114) Delay, Memory		
(115) Delay, Memory		
(116) V EFF to MAR (V Operand)	MAR	V (M1)'
(117) Read Memory (V Operand)		
(118) Delay, Memory		
(119) Delay, Memory		
(120) Delay, Memory		
(121) Word appears in RR	RR	V Operand
(122) RR to X - (V)	X	V Operand
(123) X to Acc	Acc	U Operand + V Operand = U Result 2

Operations	Register Affected	Contents
(124) Delay, Memory and Acc		
(125) Delay, Memory and Acc		
(126) U EFF to MAR (U Result)	MAR	U (M1)'
(127) Acc to X - (U)	X	U Result 2
(128) Write Memory (U Result 2)		
(129) X to RR (NOTE: START OF 3RD CYCLE)	RR	U Result 2
(130) UMR to MAD - (U)	MAD	U Modifier ''
(131) UAR to MAD - (U)	MAD	U Modifier '' + U Add = U (M2)
(132) Delay, Memory and MAD		
(133) Delay, Memory and MAD		
(134) MAD to U EFF - (U)	U EFF	U (M2)
(135) Delay, Memory		
(136) Delay, Memory		
(137) MAD to MAR (U Indirect)	MAR	U (M2)
(138) Read Memory (U Indirect) UIR to MAD - (U)	 MAD	 U Add Increment
(139) UMR to MAD - (U)	MAD	U Add Increment + U Modifier '' = U Modifier '''
(140) Delay, Memory and MAD		
(141) Delay, Memory and MAD		
(142) MAD to UMR - (U) Word appears in RR (U Direct)	UMR RR	U Modifier ''' U (M2)'
(143) RR to X - (U) UCTR to MAD - (U)	X MAD	U (M2)' U Ct - 2

Operations	Register Affected	Contents
(144) MAD - 1 to MAD	MAD	U Ct - 3
(145) Delay, MAD		
(146) Delay, MAD		
(147) Clear MAD	MAD	Clear
UJR to XTR	XTR	U Jump Increment
(148) XTR to MAD	MAD	U Jump Increment extended

Total Time, 296 ns.

Obviously, the above notation gets quite cumbersome. In the particular mechanization chosen, U Ct negative stopped the processing, requiring a start on the 3rd (abortive) cycle. A more appropriate method can be chosen, obviating the extra time; however, this will be left until some later time.

There are many cases of overlapping in this rather complicated example. For instance, 31, 75 and 76 overlapped the processing of the U address with a memory access delay; 29 and 30 overlapped a MAD delay with a memory access delay, etc. However, 94, 95 and 96 illustrate a delay due solely to MAD time and register-to-register transfer time, and not to a memory access delay. This shows that during the processing of an instruction, there are periods when delays will arise for one reason or another which will not be overlapped with anything, due to the nature of the processing involved (disregarding multi-programming concepts). Another example of this is 108, 109 and 110 where nothing can be done until the operand is retrieved from memory. It is the sequential nature of processing that engenders the delays. The extensive nature of euprosodopoeticizing seems to keep these delays within bounds, according to the particular timing assumptions chosen, of course. Of the total memory access delay of 108 ns, 60 ns were not overlapped with anything, 38 ns were overlapped with a MAD delay, and 10 ns were overlapped with an Acc delay. Also, there was a MAD delay of 14 ns that was not overlapped with anything. However, out of the total time of 296 ns, some time may be saved by having a different ending sequence than the one chosen for this example. There were 12 accesses to the high-speed memory, numbered 4, 26, 41, 51, 61, 73, 83, 97, 107, 117, 128, and 138. As can be seen, sometimes the memory was accessed in just 20 ns, sometimes a few extra ns were needed, and occasionally a large amount of time was spent in between accesses, for instance, 26 to 41, due to fetching B-boxes. Of the total of 240 ns of memory access time, about 2/3 (or a little less) were overlapped with something. It will be seen, of course, that the example could be mechanized a little differently, changing the delays and overlaps to a considerable extent. The example does illustrate, however, the possibilities arising for paralleling, at the cost of an increasing complexity in the program control unit.

In the previous example which was non-repeat, the time for essentially the same operational procedure was 150 ns. The time for doing the same process twice using the repeat mode was 296 ns, again showing that most of the extra operations involved in the repeating were overlapped and there were relatively fewer memory accesses. There will be some "break-even" point for instructions under repeated and non-repeated mode, which will probably differ for each instruction. More detailed mechanization of the individual instructions will provide better timing information.

J. INSTRUCTION REPERTOIRE

Finally, we have reached the point of presenting the instructions for Minotaur I. It will be mentioned at the outset that there will undoubtedly be many changes, additions and deletions in this list of instructions before the list is finally "frozen". The list given here is considered to be a good starting point for the bitter battles to be fought over the minute details of the individual instructions by all the many participants who will enter the arena in the future. The input-output instructions have been separated from the following list and are presented in Section K.

DATA HANDLING INSTRUCTIONS

(1) Transfer Data Positive — (U) to V

(2) Transfer Data Negative — minus (U) to (V)

(3) Transfer Data Absolute —|(U)|to V

(4) Swap U and V — (U) to V and (V) to U

(5) Load Acc and Q — (U) to Acc and (V) to Q

(6) Store Acc and Q — Acc to U and Q to V

(7) Logical Product and Shift — L(Q) (U) to Acc, shift left circular by V. This instruction can be used to pack equal sized characters into the Accumulator when used in the repeat mode.

(8) Shift and Partial Store — Shift Acc by V, and then L(Q) (Acc) to U, don't destroy Acc (it may be necessary to clear the rest of U). This instruction can be used to unpack equal sized characters from the Accumulator into a series of words, when used in the repeat mode.

(9) Load or Add U and Shift (it is a question whether this instruction belongs here or under arithmetic instructions). The shift count is given by V.

x x x x x x x

(a) Load into Acc - O
Load into Q - 1

(b) Don't Load Operand - O
Load Operand from U - 1

(c) Don't Add Operand - O
Add Operand from U - 1

(d) Don't Store Result - O
Store Result in U after Shift - 1

(e) Shift Left - O
Shift Right - 1

(f) Circular Shift - O
Non-circular Shift - 1

(g) Short Shift (Only Acc shifted) - O
Long Shift (Acc and Q shifted
as one long register) - 1

It will be noticed that with the appropriate choice of bits in the op symbol, it will be possible to accomplish nothing. It will not be possible to have both (b) and (c) "1" in the same instruction. Also, if (a) is "1", there doesn't seem to be much point in having (g) "O".

(10) Load or Subtract U and Shift. The shift count is given by V.

x x x x x x x

(a) Load into Acc - O
Load into Q - 1

(b) Don't Load Operand - O
Load Complement of U - 1

(c) Don't Subtract Operand - O
Subtract Operand from U - 1
(Not valid if (a) is 1)

(d) Don't Store Result after Shift - O
Store Result in U after Shift - 1

(e) Shift Left - O
Shift Right - 1

(f) Circular Shift - O
Non-circular Shift - 1

(g) Short Shift - O
Long Shift - 1

It may be advisable to provide the option of storing the result in (9) and (10) through a mask. This would require the addition of a 60-bit mask register to the program control. It is doubtful that the function thus provided would be worth the cost (by present thinking).

Both (9) and (10) above use some bits of the op symbol to provide the necessary optional functions. This cuts down on the choice of the remainder of the op symbols for the other instructions, but it is believed that there are still enough for the remainder of the instructions.

(11) Count "1" Bits — The purpose of this instruction is to determine how many and which bits in a word are "1" (for instance the I/O interrupt register). Given the word in Q, it makes a list of those bits which are "1"; the final U address can be used to determine how many bits were "1". It does this in the following way: Don't clear Acc; if Q(60) is "1", then add one to Acc, transfer Acc (15-0) to (U(15-0)), add one to U Add so the next time Acc (15-0) is stored, it will be stored in (U+1 (15-0)), subtract one from V, and shift Q left one place, (circularly). If Q(60) is "0", add one to Acc, subtract one from V, and shift Q one place left. The instruction stops when V is zero (or negative), thereby enabling parts of words to be tested. It may be desirable to transfer the U Address to Acc (39-24) at the completion of the instruction. This instruction is not repeatable. It might be that the form of this instruction will change, depending on the mechanization of several other instructions (shift, add, etc.) such that in the non-repeat mode, the instruction will test only one bit; but in the repeat mode, it will perform the functions described here. For the present, it will be left as is.

ADDRESS HANDLING INSTRUCTIONS

(12) Transfer U Address to U Address — U(U) to V (U), i.e. (U(17-0)) to (V(17-0))

(13) Transfer U Address to V Address — U(U) to V(V)

(14) Transfer V Address to U Address — U(V) to V(U)

(15) Transfer V Address to V Address — U(V) to V(V)

In the above four transfers, all 18 bits participate in the transfer. (It is possible, using the above four instructions twice, to Transfer U EFF into V. The details are left as an exercise for the reader).

(16) Add Modifiers — (U)(15-0) + (V)(15-0) to U(15-0). When used in repeat mode, this instruction will sum modifiers in B-boxes, permitting a sum of modifiers to be used to modify a single address.

(17) Add Increments — (U) (39-24) + (V) (39-24) to U (39-24). When used in repeat mode, this instruction will sum increments in B-boxes.

(18) Add Counts — (U) (57-42) + (V) (57-42) to U (57-42). When used in repeat mode, this instruction will sum counts in B-boxes.

(19) Add Increment to Modifier — (U) (15-0) + (U) (39-24) to U (15-0) (probably not repeatable). This instruction performs the same function as part of the repeat sequence, used in loop control.

(20) Add Increment to Modifier, Count — (U) (15-0) + (U) (39-24) to U (15-0) and (U) (57-42) - 1 to U (57-42) (probably not repeatable).

(21) Add Increment to Modifier, Count, Self-Relative Jump on Negative — (U) (15-0) + (U) (39-24) to U (15-0), (U) (57-42) - 1 to U (57-42). If Negative, Transfer U Jump Increment Extended to MAD (probably not repeatable).

ARITHMETIC INSTRUCTIONS

(22) Multiply — U x V, Leave Result in Acc and Q

(23) Multiply and Add — Acc (i) + U x V, Leave Result in Acc and Q

(24) Divide — Acc and Q by U, Remainder in Acc, Quotient to V

(25) Add, Subtract and Transfer

(a) Clear Acc - 0
Don't clear Acc - 1

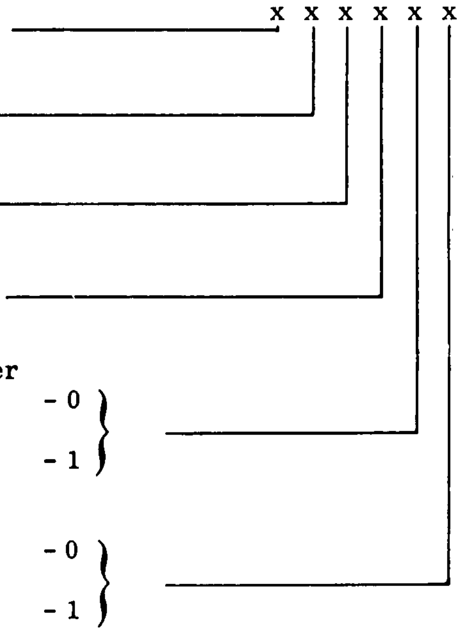
(b) Don't Add U - 0
Add U - 1

(c) Don't Add V - 0
Add V - 1

(d) Don't Subtract V - 0
Subtract V - 1

(e) Don't Store Result in U after the Operation - 0
Store the Result in U after the Operation - 1

(f) Don't Store the Result in V after the Operation - 0
Store the Result in V after the Operation - 1



It may not be possible to have both (e) and (f) "1" in the same instruction.
It is not possible to have both (c) and (d) "1" in the same instruction.

LOGICAL INSTRUCTIONS

All 16 logical operations will be provided on full-word operands U and V, with the result replacing the full word at U. Q, and possibly Acc, will be used in the operation of the instructions. No names will be given to these instructions at the present time.

(26) to (41) The 16 logical operations on two operands are as follows:

U	V	26	27	28	29	30	31	32	33	34	35	36	37	38	39	40	41
0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1
0	1	0	0	0	0	1	1	1	1	0	0	0	0	1	1	1	1
1	0	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1
1	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1

DECISION AND CONTROL INSTRUCTIONS

There are a large number of these instructions, as befits a data processor. An attempt has been made to make the decision instructions symmetrical in the jump condition, so that the programmer has his choice at each decision point. This has not been completely successful; however, most of the functions are doubled.

(42) Unconditional Jump to U

(43) Unconditional Jump to V

(44) Return Jump — PAR + 1 to U, Jump to V

(45) Unconditional Stop after Euprosodopoeticizing

(46) If Acc Equals Zero, Jump to U; if Acc is not Zero, Jump to V

(47) If Acc is Greater than or Equal to Zero, Jump to U; if Acc is Less than Zero, Jump to V

(48) If Acc is Greater than Zero, Jump to U; If Acc Equals Zero, NI (Next Instruction)
If Acc is Less than Zero, Jump to V

(49) If Acc (60) is "1", Left Circular Shift Acc by One and Jump to U;
If Acc (60) is "0", Left Circular Shift Acc by One and Jump to V.

(50) If Q Equals Zero, Jump to U; If Q is not Equal to Zero, Jump to V

(51) If Q is Greater than or Equal to Zero, Jump to U; if Q is Less than Zero, Jump to V.

(52) If Q is Greater than Zero, Jump to U; If Q is Equal to Zero, NI; If Q is less than Zero, Jump to V.

(53) If Q(60) is "1", Jump to U; If Q(60) is "0", Jump to V; In Either Case, Left Circular Shift Q by One.

(54) If Acc is Greater than Q, Jump to U; If Acc is Equal to Q, NI; If Acc is Less than Q, Jump to V.

(55) If (U) Equals Zero, Jump to V; otherwise, NI

(56) If (U) Doesn't Equal Zero, Jump to V; otherwise, NI

(57) If (U) is Greater than or Equal to Zero, Jump to V; otherwise, NI

(58) If (U) is Less than Zero, Jump to V; otherwise, NI

(59) If (U) Equals Acc, Jump to V; otherwise, NI

(60) If (U) Doesn't Equal Acc, Jump to V; otherwise, NI

(61) If (U) is Greater than or Equal to Acc, Jump to V; otherwise, NI

(62) If (U) is Less than the Acc, Jump to V; otherwise, NI

(63) If (U) Equals Q, Jump to V; otherwise, NI

(64) If (U) Doesn't Equal Q, Jump to V; otherwise, NI

(65) If (U) is Greater than or Equal to Q, Jump to V; otherwise, NI

(66) If (U) is Less than Q, Jump to V; otherwise, NI

(67) If L(Q)(U) Equals Zero, Jump to V; otherwise, NI

(68) If L(Q)(U) is not Equal to Zero, Jump to V; otherwise, NI

(69) If L(Q)(U) is Equal to Acc, Jump to V; otherwise, NI

(70) If L(Q)(U) is not Equal to Acc, Jump to V; otherwise, NI

(71) If L(Q)(U) Equals L(Q) (Acc), Jump to V; otherwise, NI

(72) If L(Q)(U) is not Equal to L(Q) (Acc), Jump to V; otherwise, NI

(73) If U (the address, not the contents of the address) Equals B(U) (15-0), Jump to V; otherwise, NI — U is not B-boxable. The purpose of this instruction is to give an immediate compare on the modifier of any B-box. It does this by forcing U to be an immediate address, and compares it against the modifier in B-box bit positions 15-0.

(74) If U is not Equal to B(U) (15-0), Jump to V; otherwise, NI — Same as 69, except the jump is the other way.

(75) Index Jump 1 — If (U) - 1 is greater than or equal to Zero, store (U) - 1 and jump to V. If (U) - 1 is less than zero, NI

(76) Index Jump 2 — If (U) - 1 is greater than or equal to zero, store (U) - 1 and NI. If (U) - 1 is less than zero, jump to V.

(77) Reverse Index Jump 1 — If (U) + 1 overflows, NI. If (U) + 1 doesn't overflow, store (U) + 1 and jump to V.

(78) Reverse Index Jump 2 — If (U) + 1 overflows, jump to V. If (U) + 1 doesn't overflow, store (U)+ 1 and NI

(79) Jump Register Index Jump 1 — If (U) - 1 is greater than or equal to zero, store (U) - 1 and NI. If it is less than zero, transfer the contents of the jump register to PAR. This permits control of loops to remain in a subsidiary routine, while controlling the place jumped to in an executive routine.

(80) Jump Register Index Jump 2 — If (U) + 1 overflows, transfer Jump Register to PAR; otherwise, store (U) +1 and NI

NOTE: V is not used in either of the above two instructions.

(81) Selective Stop, Jump, OR — There will be 16 three-position switches on the console marked, "Stop", "Jump", "Ignore". The conjunction of a bit in U and the switch will cause that function to be executed. In case of conflicts, a priority system will be put into effect. If there are multiple conjunctions in either stop or jump positions, the decision will be based on the "logical Or" of the patterns. For example, this means that if two switches were set to jump, then if <u>either</u> of these bits in U were "1", the jump would be executed.

(82) Selective Stop, Jump, AND — This is the same as (81) except the functions are "ANDED". For instance, if two switches were set to jump, then <u>both</u> of those bit positions in U would have to be "1" in order for the jump to take place.

(83) Sense Interrupt Register — If <u>any</u> bit in the IR is "1", jump to U. If <u>all</u> the bits in IR are "0", jump to V.

(84) Alternate FF Test 1 — There will be a FF in the control portion of the computer used by this instruction and the one below in the following manner: if the last time this FF was tested, U was transferred to PAR, this time transfer V to PAR. If last time, V was transferred to PAR, then this time transfer U to PAR.

(85) Alternate FF Test 2 — If last time this FF was tested U was transferred to PAR, transfer U to PAR again. If last time this FF was tested V was transferred to PAR, then transfer V to PAR again.

NOTE: The above two instructions can be used together for various decision paths through a network of program control logic.

(86) Program Control FF Test — There will be a series of FF that can be set, reset, triggered, and tested by this instruction in order to control the program. Most probably, the number of FF's will be 16. They are set, reset or triggered and tested with this single instruction. The instruction consists of two parts, "test" and "do". The bits for controlling the "test" and "do" functions will be contained in the op symbol. The FF's to be tested will be identified by the bits in U.

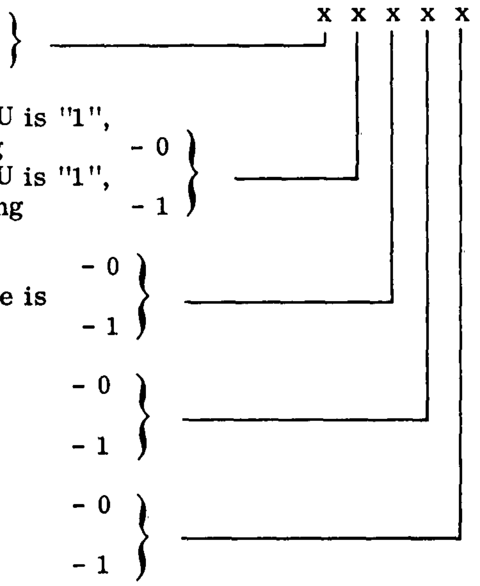
(a) Jump to V if Set - 0
Jump to V if Reset - 1

(b) If more than one bit in U is "1", OR the bits for testing - 0
If more than one bit in U is "1", AND the bits for testing - 1

(c) Don't set any FF - 0
Set each FF where there is "1" in U - 1

(d) Don't reset any FF - 0
Reset each FF where there is "1" in U - 1

(e) Don't trigger any FF - 0
Trigger each FF where there is "1" in U - 1



This instruction works somewhat as follows: assume that (a) is "1", (e) is "1", and that there are two "1" bits in U, namely FF's x and y. Then, FF x and y will be tested, and if either is reset, there will be a jump to V. In any case, the FF's corresponding to bit positions x and y will be triggered after the test.

(87) Trace Jump, Normal

(88) Trace Jump, Return

(89) Execute — Execute the instruction located at U EFF. If any attempt is made to change sequence, do a return jump to some fixed memory locations instead.

K. INPUT-OUTPUT

The most difficult area has been left to last. Unfortunately this is also the most turgid area, since the specific details of all the possible input-output devices are unknown at the present time. However, the generalized outlines of a scheme for controlling input-output functions and specific device types with their control functions can be listed. This precludes the listing of specific instructions, as was done in the previous section.

The general form of the input-output instructions will be given, together with a listing of the devices to be controlled, the types of control, and an outline of the buffering and interrupt schemes for mechanizing the control. Details will be given wherever possible.

There will be three types of input-output instructions, which are usable at the programmer's option. Not all three will be available for each device, due to the nature of the devices, but flexibility will be maintained where possible. The three types are: non-parallel, "time-shared" parallel, and "program" parallel, each of which are defined more fully below:

1. Non-Parallel

This type of instruction will hold the machine up completely until the operation is completed. (The cries of the wounded can probably be heard in the vicinity of Alpha Centauri. To those I say, "You have never tried for several hours, in vain, to bootstrap a program into a machine that was not operating 100% properly.") This type of I/O should be used for loading programs, maintenance routines, or for cases where the memory has been cleared, or nothing else can be done until more information is received in the memory. It is hoped that this mode will be used only with the higher-speed devices, for instance, megacycle magnetic tapes. However, facility for using low-speed devices will also be provided. It is the opinion of the authors that regardless of the elaborateness of any I/O scheme, or the safety checks built into it, nothing will prevent some programmers from using the computer incorrectly. In any case, the use of this type of I/O instruction will waste computer time by having it wait for much slower devices to operate. The waste will be greater in this computer because of its greatly increased speed. It is hoped that this waste will be kept to a minimum.

2. Time-Shared Parallel

This mode permits a degree of simultaneity by time sharing the memory. An examination of Figures II-5, II-6 and II-7 will be helpful here because they contain the format of an I/O instruction and of an I/O control word before and after euprosodopoeticizing. In this mode, after an I/O instruction has been euprosodopoeticized, the control word is read into the X register and ICR, euprosodopoeticized and then written into some standard memory location which differs for each device. There will be a correspondence, then, between the U addresses of the I/O instructions and the standard locations in the memory. The device will be started, appropriately, and then the next instruction will be euprosodopoeticized. The "Main Frame" will be interrupted for several cycles every time the device has filled or emptied its buffer, in the case of inputs or outputs, respectively. During this interruption, another word will be read into or out of the memory, addresses will be advanced and checked for termination, etc.; and then, the regular processing will continue. It will thus be possible to control a number of devices simultaneously.

The error/end condition address can be used at the programmer's option to jump to any location within the first 4096 locations at the termination of an instruction, or in case of an error (parity error, for instance). Unfortunately, the end condition address must be absolute, which makes it rather difficult for generalized I/O routines. It will be noticed, however, that the beginning and ending data addresses can be B-boxed and indirectly addressed. If the programmer chooses not to exercise the end address option, he may test the condition of the various devices with control-type decision instructions. Some instructions will not require the control word (i.e., rewind tape "k" to the beginning), in which case the V address is not used, nor is a control word required. Some examples of instructions of this type and of the type preceding are: read the next block of words from magnetic tape "k" into memory, starting at address "j" and ending at address "j + b" (if a conflict exists between the block length and the address given for termination, there will be some indication given); write the block of words starting at "j" and ending at "j + b" to magnetic tape "k"; punch the block of words starting at "j" and ending at "j + b" on paper tape; etc.

This system of I/O control requires, among other things, character-to-word buffers, having differing sizes for each different device. On the output side, word-to-character buffers are required and perhaps not the same as the ones on the input side. A more complete explanation will be given below.

A series of control and priority FF will be required in the control part of the computer, and a block of memory with special address-generating logic for the control words. Also, some internal sequencing is required for advancing addresses, etc.; but it is hoped that the subcommands already provided for other instructions will suffice.

The buffer system will consist of the following: there will be one tunnel diode register, (in addition to the I/O exchange) of full-word length; and a number of full-word length shift registers attached to the tunnel diode register, fabricated from high-speed transistors. Probably, eight of these will suffice. Attached to each shift register will be a number of staticizers, not all of the same size, that will accept or feed information directly from or to the I/O devices. Some of these staticizers will be able to be connected to more than one shift register. In this way, more than one device per shift register can be operating simultaneously. It may also be possible to arrange a few staticizers being connected directly to the tunnel diode register (for megacycle magnetic tapes, for instance). The staticizers will come in sizes of 6, 7, 12 and 24 bits, and others as required. It seems likely that there will be extensive switching facilities between the shift registers and the staticizers. There will be less extensive switching between the tunnel diode register and the shift registers. The control and the information signals will have to be switched. The switching network between the tunnel diode register and the shift registers may have to be tunnel diodes. The network between the shift registers and the staticizers can be transistors.

3. Program Parallel.

This system is somewhat similar to that used on BOGART, in that an instruction is necessary for each signal or character sent to an external device. Therefore, the programmer must time out his program for those devices which operate asynchronously, in real-time, and cannot be stopped between characters or frames; for example, magnetic tapes, card readers with only buffering for one line and that can't be stopped between lines; etc. The advantage is that, relative to the non-parallel mode, very little "main frame" time is used in information transfers to and from slow-speed devices.

However, the programmer usually has quite a time insuring that no information is lost from certain devices. The net result is that the effective lost time is greater than appears at first to be the case. The appropriate sense instructions will help alleviate this situation. It may be that more than one I/O buffer is required for this system, but it will be left open for the present. This system, of course, doesn't use the end condition address, and probably, dispenses with the control word altogether. This makes the V address in the I/O instruction the data address. An example of this type of instruction is — "Read the next character from the paper tape buffer into location "j". Another example would be — "Read the next character from the paper tape into the paper tape buffer".

The three systems outlined above will be called "A", "B" and "C" for the present. There will be some redundancy among the various options listed above, in which case, the programmer will be able to choose the method that best suits his particular needs at the moment. As has been mentioned, the I/O scheme is quite extensive, and will require a lot of hardware. It is felt, however, that it will be worth the cost.

Following is a listing of the various devices proposed at the present time, and the type of control needed for each of the devices, given in lieu of an instruction list. The letters in parenthesis refer to A, B, and C listed above as the three schemes proposed.

(1) Magnetic Tape

(a) Read a Block in the Forward Direction (A), (B)

(b) Read a Block in the Reverse Direction (A), (B)

(c) Read a Character in the Forward Direction (C)

(d) Read a Character in the Reverse Direction (C)

(e) Write a Block in the Forward Direction (A), (B)

(f) Write a Character in the Forward Direction (C)

(g) Rewind a Tape to Beginning

(h) Unwind a Tape to "End Tape Warning"

(i) Rewind a Tape a Specific Number of Blocks

(j) Unwind a Tape a Specific Number of Blocks

(k) Sense Beginning of Tape

(l) Sense "End Tape Warning"

(m) Sense Any of Following: No Tape, Moving Forward, Moving Reverse, Motion or No Motion, Tape Operable

(2) Paper Tape

(a) Start Motor (?)

(b) Read a Character from PPT to Buffer (C)

(c) Read a Character from PPT Buffer into the I/O Exchange (C)

(d) Write a character from Exchange to PPT Buffer, and Punch when Possible (C)

(e) Read Next Block of Characters from PPT into Memory (presupposes end of block marker or odd parity on tape) (A), (B)

(f) Write a Block of Characters from the Memory to PPT (A), (B)

(3) Punched Cards

There will be some means of reading and punching cards; details of this means are left open for the present.

(4) Typewriter

(a) Write a Character from Memory (?) to Typewriter (C)

(b) Read a Character from Typewriter to Memory (C)

(c) Write a Block of Characters to Typewriter (A), (B)

(d) Read a Block of Characters from Typewriter to Memory (A), (B)

(e) Sense Typewriter Busy

(5) Line Printer

(a) Print a Line on Line Printer (A), (B)

(b) Do one of the following functions: vertical tab, page change, line shift a certain number of lines, line shift according to a control loop on the printer (A), (B), (C)

(c) Print a Character in Next Position of Line Printer (C)

(6) Point Plotters

(a) Write to Point Plotter, i.e., send to plotter X and Y coordinates and a symbol (12 bits)

(b) Sense if Plotter is Busy

(c) Advance to Next Frame

(d) Sense if Advance has Completed

(7) Pictorial Matter Readers — These operate as follows: give the reader the X and Y coordinate of the point desired to be read; and in less than 10 us, a reading in grey scale will return (12 bits).

(a) Give X and Y Coordinates, Symbol Returned to I/O Exchange Register (B)

(b) Unwind N Frames (C), (B?)

(c) Rewind N Frames (C), (B?)

(d) Sense End of Unwind or Rewind (C)

(8) Real-Time Clock.

(a) Zero Clock (A)

(b) Read Clock (A)

(c) Set Clock (A)

(d) Start Clock (A)

(e) Stop Clock (A)

(f) Jump on Overflow (B)

(9) Shaft Positioners — There will be a digital to analog converter attached to a multiplexer which can have up to 64 motors attached to it. Each motor has a potentiometer sensing its position. Each potentiometer will be connected through another multiplexer to an analog to digital converter and back to the computer. Control signals can be sent through the digital to analog converter and multiplexer to the motors in order to change their position in either a forward or reverse direction, depending on the number sent out. The speed of the motors will be relatively proportional to the difference between their present position, and the requested position. The motors can thus be moved slowly by sending out small changes in position over a long interval. Each of the two multiplexers is independent of the other. The computer sends out and receives control signals from the multiplexers, 6 bits for connections, and 12 bits for "delta" voltages. Also, the present positions of the multiplexers can be read.

(a) Set Multiplexer (Motors)

(b) Set Multiplexer (Potentiometer)

(c) Read Multiplexer (M) or (P) — i.e., to which of 64 positions is Multiplexer set

(d) Sense if Multiplexer (M) or (P) is Busy

(e) Write "Delta" Voltage to Motor to which Multiplexer is Set (12 Bits)

(f) Read Potentiometer to which Multiplexer is Set (12 Bits)

There may turn out to be a need for some more sensing instructions, but the question will be left open for the present.

L. CONSOLE

Nothing so far has been said of the console. It may turn out that an interpretive console is required, due to the difficulty of a tunnel diode-transistor interface. In any case, some amount of buffering will be required.

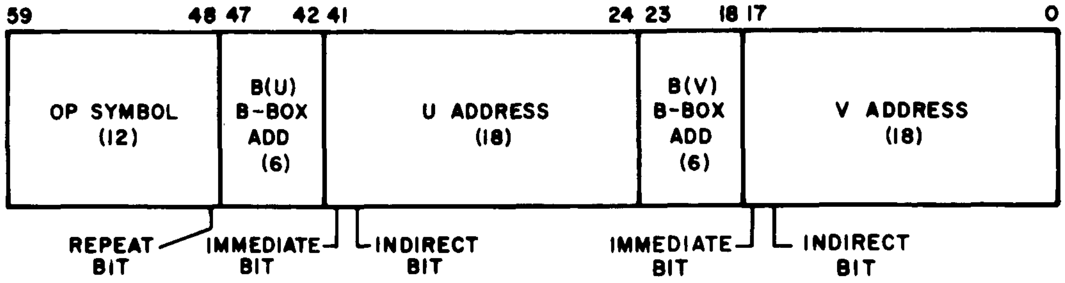


Figure II-1. Instruction Word

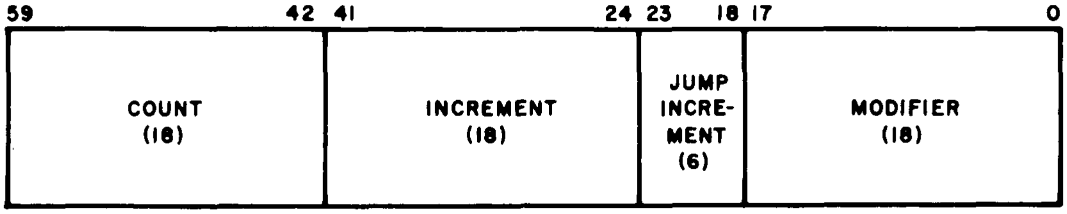
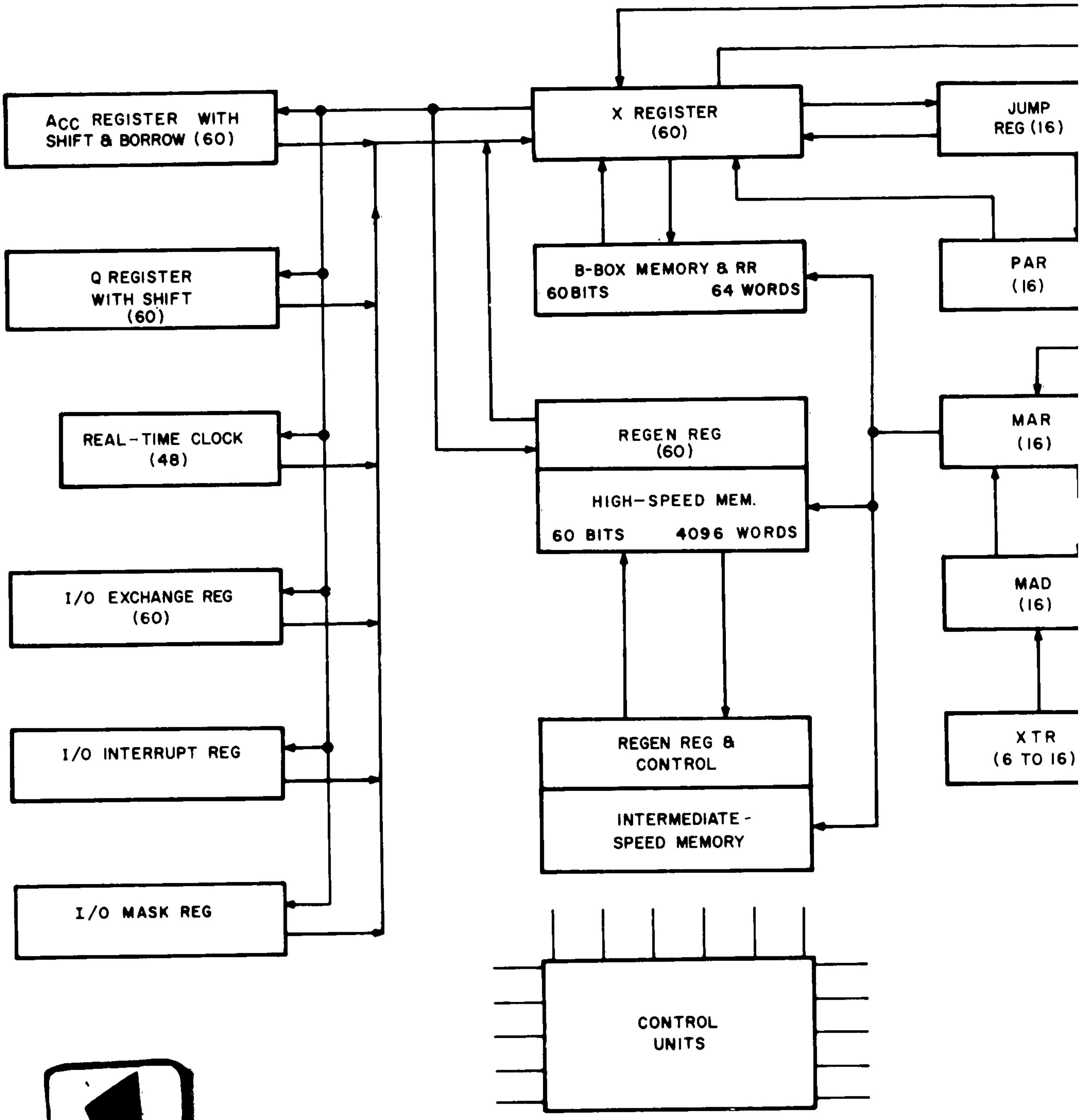


Figure II-2. B-Box Word


ACC REGISTER WITH SHIFT & BORROW (60)
Q REGISTER WITH SHIFT (60)
REAL-TIME CLOCK (48)
I/O EXCHANGE REG (60)
I/O INTERRUPT REG
I/O MASK REG
X REGISTER (60)
JUMP REG (16)
B-BOX MEMORY & RR
60 BITS
64 WORDS
PAR (16)
REGEN REG (60)
HIGH-SPEED MEM.
60 BITS
4096 WORDS
MAR (16)
MAD (16)
XTR (6 TO 16)
REGEN REG & CONTROL
INTERMEDIATE - SPEED MEMORY
CONTROL UNITS

1

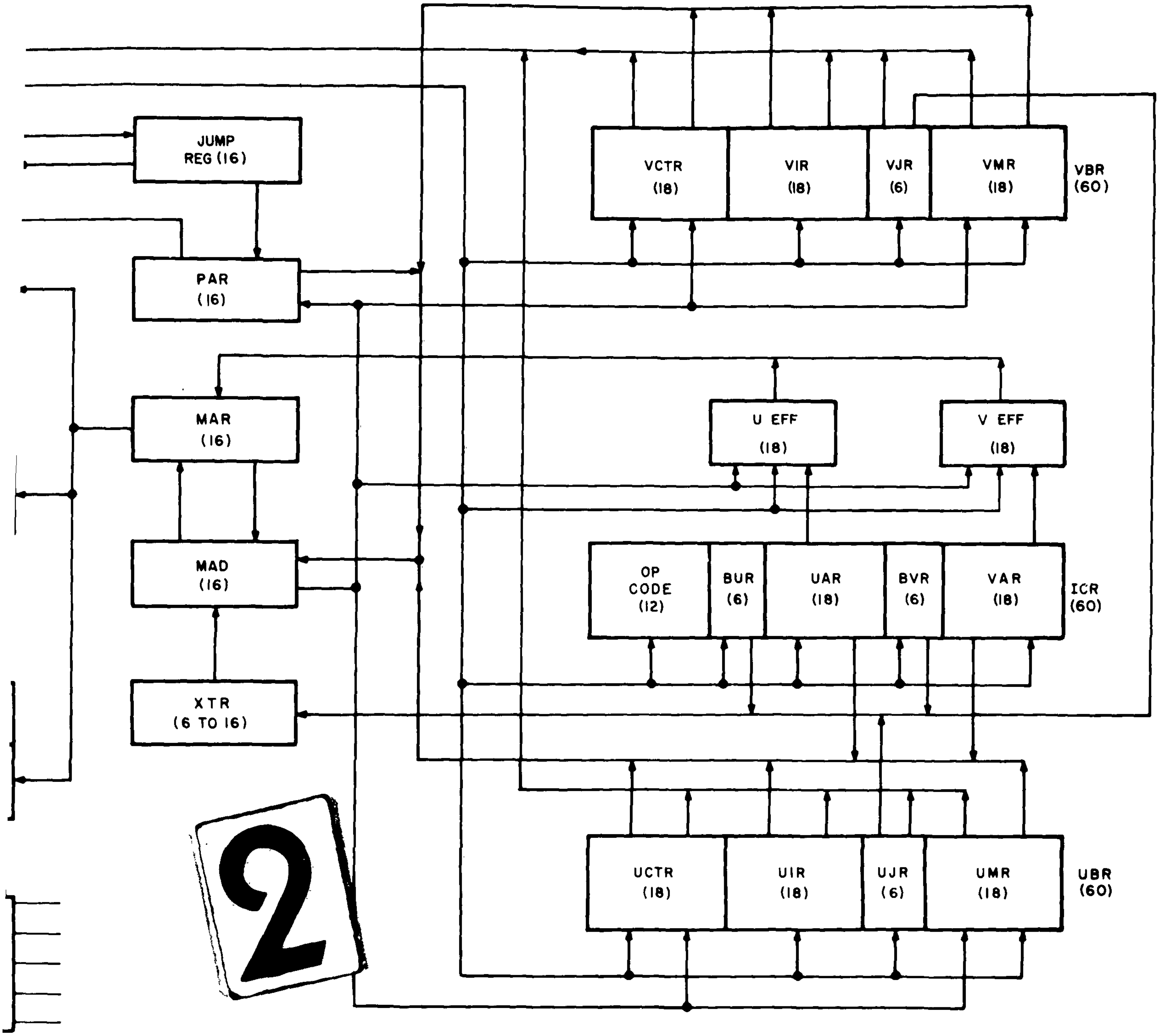
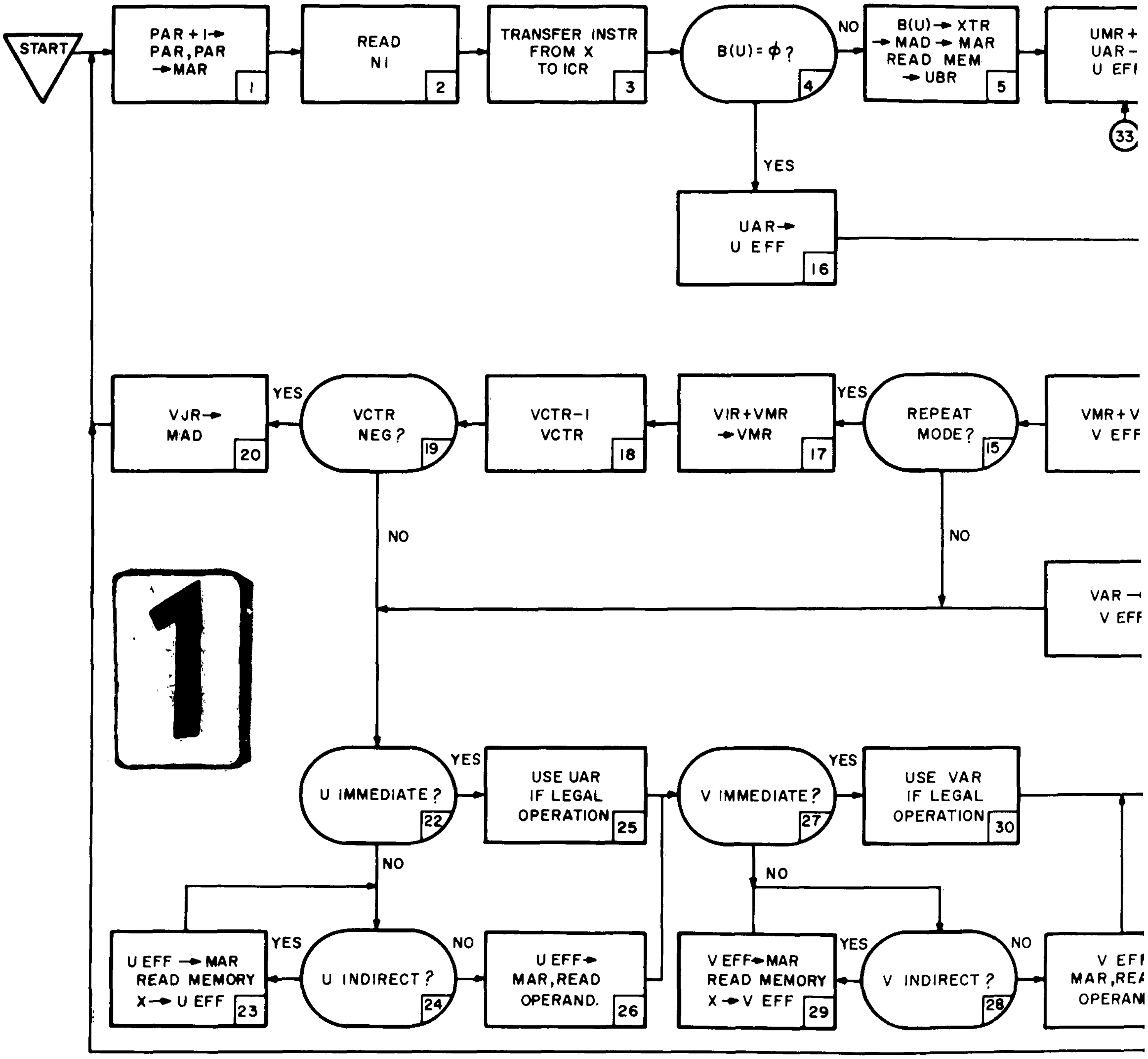


Figure II-3. Control Registers, Minotaur I


START
PAR+I→ PAR, PAR →MAR 1
READ NI 2
TRANSFER INSTR FROM X TO ICR 3
B(U)=ϕ? 4
NO
B(U)→XTR →MAD→MAR READ MEM →UBR 5
UMR+ UAR– U EFI
33
YES
UAR→ U EFF 16
VJR→ MAD 20
YES
VCTR NEG? 19
VCTR–I VCTR 18
VIR+VMR →VMR 17
YES
REPEAT MODE? 15
VMR+V V EFF
NO
NO
VAR → V EFF
1
U IMMEDIATE? 22
YES
USE UAR IF LEGAL OPERATION 25
V IMMEDIATE? 27
YES
USE VAR IF LEGAL OPERATION 30
NO
NO
U EFF →MAR READ MEMORY X→U EFF 23
YES
U INDIRECT? 24
NO
U EFF→ MAR, READ OPERAND. 26
V EFF←MAR READ MEMORY X→V EFF 29
YES
V INDIRECT? 28
NO
V EFI MAR, REA OPERAN

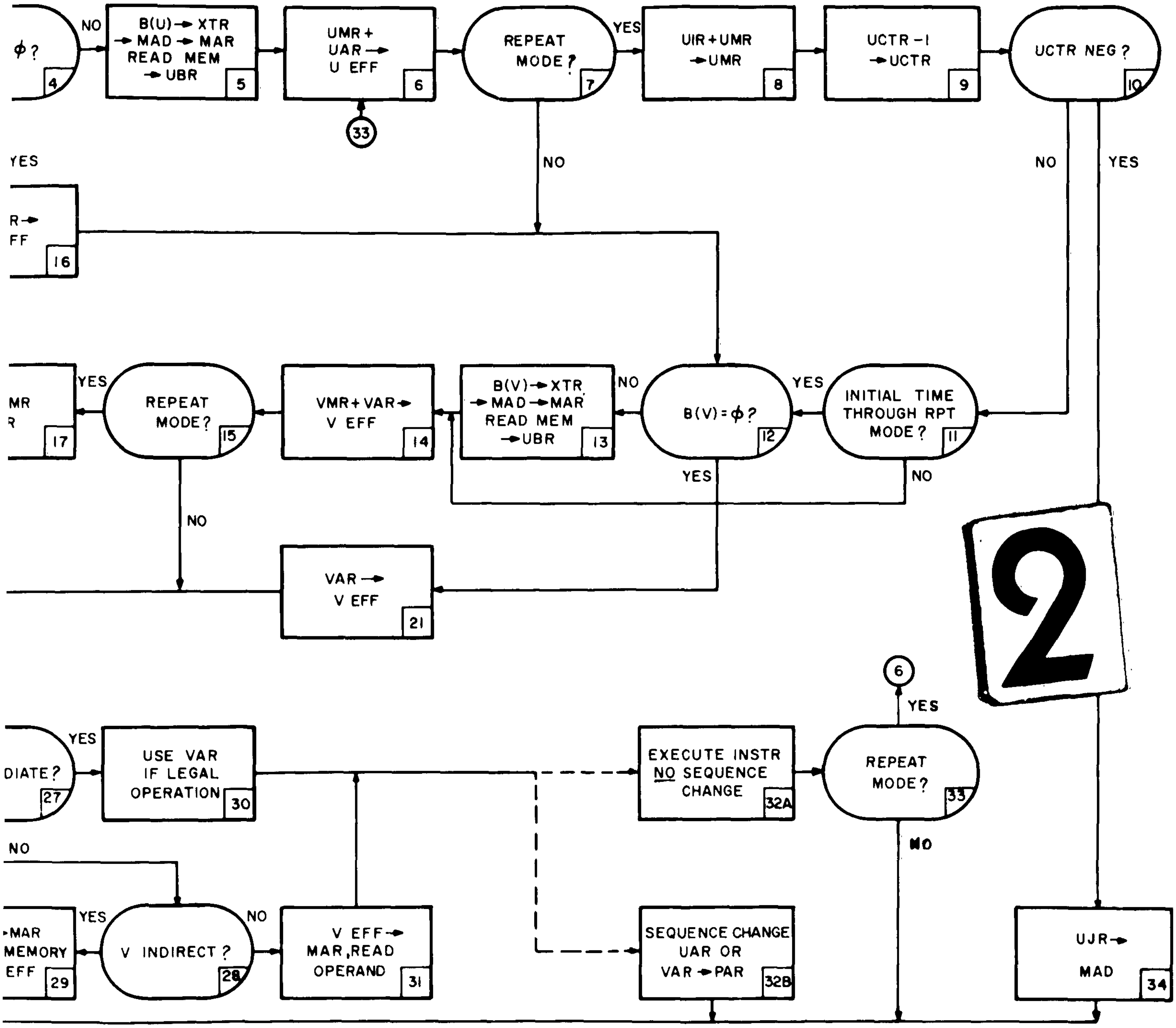


Figure II-4. Euprosodopoesis — Minotaur I

59 48	47 42	41 24	23 18	17 0
OP SYMBOL	B(U)	I/O DEVICE ADDRESS	B(V)	CONTROL WORD ADDRESS OR DATA WORD ADDRESS OR PARAMETER

Figure II-5. Input-Output Instruction Word

59 48	47 42	41 24	23 18	17 0
ERROR/END CONDITION ADDRESS	B(U)	INITIAL DATA ADDRESS	B(V)	TERMINAL DATA ADDRESS

Figure II-6. Initial I O Control Word

59 48	47 42	41 24	23 18	17 0
ERROR/END CONDITION ADDRESS	B(U)	EFFECTIVE INITIAL DATA ADDRESS	B(V)	EFFECTIVE TERMINAL DATA ADDRESS

Figure II-7. I O Control Word as Stored in Standard Location